Analyzing the Lending Club Case Study

Importing the Libraries required for EDA

In [213]:
```
#import the required Libararies
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt, seaborn as sns

#adjusting the rows and columns display
pd.set_option('display.max_columns',111)
pd.set_option('display.max_rows',111)
```

1. Reading the Input Data from the File

In [214]:
```
#Reading the loan data in pandas
file_path = 'C:/Users/SRSRE/Downloads/Loan DataSet/loan.csv'
loan_df = pd.read_csv(file_path, low_memory = False)
loan_df.head()
```

Out[214]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment
0	1077501	1296599	5000	5000	4975.0	36 months	10.65%	162.87
1	1077430	1314167	2500	2500	2500.0	60 months	15.27%	59.83
2	1077175	1313524	2400	2400	2400.0	36 months	15.96%	84.33
3	1076863	1277178	10000	10000	10000.0	36 months	13.49%	339.31
4	1075358	1311748	3000	3000	3000.0	60 months	12.69%	67.79

In [216]:
```
#getting the column informations
loan_df.dtypes
```

2. Understanding structure of the Data

In [215]:
```
#getting the dataframe dimensions
loan_df.shape
```

Out[215]:
```
(39717, 111)
```

```
num_bc_tl                   float64
num_il_tl                   float64
num_op_rev_tl               float64
num_rev_accts               float64
num_rev_tl_bal_gt_0         float64
num_sats                    float64
num_tl_120dpd_2m            float64
num_tl_30dpd                float64
num_tl_90g_dpd_24m          float64
num_tl_op_past_12m          float64
pct_tl_nvr_dlq              float64
percent_bc_gt_75            float64
pub_rec_bankruptcies        float64
```

In [216]:
```
#getting the column informations
loan_df.dtypes
```

```
num_bc_sats                 float64
num_bc_tl                   float64
num_il_tl                   float64
num_op_rev_tl               float64
num_rev_accts               float64
num_rev_tl_bal_gt_0         float64
num_sats                    float64
num_tl_120dpd_2m            float64
num_tl_30dpd                float64
num_tl_90g_dpd_24m          float64
num_tl_op_past_12m          float64
pct_tl_nvr_dlq              float64
percent_bc_gt_75            float64
pub_rec_bankruptcies        float64
tax_liens                   float64
tot_hi_cred_lim             float64
total_bal_ex_mort           float64
total_bc_limit              float64
total_il_high_credit_limit  float64
dtype: object
```

2. Understanding structure of the Data

In [215]:
```
#getting the dataframe dimensions
loan_df.shape
```

Out[215]:
```
(39717, 111)
```

In [217]:
```
#basic info of the data frame
loan_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Columns: 111 entries, id to total_il_high_credit_limit
dtypes: float64(74), int64(13), object(24)
memory usage: 33.6+ MB
```

In [218]:
```
#Getting basic statistical details of the data frame
loan_df.describe()
```

Out[218]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	installment
count	3.971700e+04	3.971700e+04	39717.000000	39717.000000	39717.000000	39717.000000
mean	6.831319e+05	8.504636e+05	11219.443815	10947.713196	10397.448868	324.561922
std	2.106941e+05	2.656783e+05	7456.670694	7187.238670	7128.450439	208.874874
min	5.473400e+04	7.069900e+04	500.000000	500.000000	0.000000	15.690000
25%	5.162210e+05	6.667800e+05	5500.000000	5400.000000	5000.000000	167.020000
50%	6.656650e+05	8.508120e+05	10000.000000	9600.000000	8975.000000	280.220000
75%	8.377550e+05	1.047339e+06	15000.000000	15000.000000	14400.000000	430.780000
max	1.077501e+06	1.314167e+06	35000.000000	35000.000000	35000.000000	1305.190000

3. Data Quality Check and Missing Values

3.1 Percentage of missing values for columns and rows

In [219]:
```
cols = pd.DataFrame(loan_df.isnull().mean().round(4) * 100, columns = ['percer
print(cols)
```

```
delinq_2yrs                 0.00
total_pymnt                 0.00
total_rec_int               0.00
total_rec_late_fee          0.00
recoveries                  0.00
collection_recovery_fee     0.00
last_pymnt_amnt             0.00
policy_code                 0.00
application_type            0.00
acc_now_delinq              0.00
delinq_amnt                 0.00
total_pymnt_inv             0.00
dti                         0.00
```

In [219]:
```
cols = pd.DataFrame(loan_df.isnull().mean().round(4) * 100, columns = ['percer
print(cols)
```

```
delinq_2yrs                                0.00
total_pymnt                                0.00
total_rec_int                              0.00
total_rec_late_fee                         0.00
recoveries                                 0.00
collection_recovery_fee                    0.00
last_pymnt_amnt                            0.00
policy_code                                0.00
application_type                           0.00
acc_now_delinq                             0.00
delinq_amnt                                0.00
total_pymnt_inv                            0.00
dti                                        0.00
total_rec_prncp                            0.00
zip_code                                   0.00
member_id                                  0.00
loan_amnt                                  0.00
addr_state                                 0.00
funded_amnt_inv                            0.00
term                                       0.00
```

In [220]:
```
#summary of missing values associated with columns
print(str(round(100.0 * cols[cols['percentage_missing_value']==0].count()/len(
print(str(round(100.0 * cols[(cols['percentage_missing_value']>0) & (cols['per
print(str(round(100.0 * cols[(cols['percentage_missing_value']>10) & (cols['pe
print(str(round(100.0 * cols[cols['percentage_missing_value']>50].count()/len(
```

```
percentage_missing_value    38.74
dtype: float64% columns have no missing value
percentage_missing_value    9.01
dtype: float64% columns have missing value betwee 0-10%
percentage_missing_value    0.9
dtype: float64% columns have missing value betwee 10-50%
percentage_missing_value    51.35
dtype: float64% columns have more than 50% missing value
```

In [221]:
```
#checking row-wise null percentages
row_null = pd.DataFrame(loan_df.isnull().sum(axis =1),columns = ['num_missing_
row_null
```

	num_missing_value
0	58
1	57
2	59
3	56
4	55
...	...
39712	59
39713	59
39714	61
39715	61
39716	59

39717 rows × 1 columns

In [223]:
```
#checking the datafrme again afer removing the columns where >50% the values a
null_values_per_column = loan_df.isnull().sum()
print(null_values_per_column)
```

```
id                          0
member_id                   0
loan_amnt                   0
funded_amnt                 0
funded_amnt_inv             0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
home_ownership              0
annual_inc                  0
```

3.2 Removing the columns with high percentage of missing values (>50%)

In [222]:
```
#removing columns where we have>50% of the values are null
threshold = 0.5
percentage_null_values = (loan_df.isnull().mean() * 100).round(2)
columns_to_remove = percentage_null_values[percentage_null_values > threshold]
loan_df= loan_df.drop(columns=columns_to_remove)
```

In [223]:
```
#checking the datafrme again afer removing the columns where >50% the values a
null_values_per_column = loan_df.isnull().sum()
print(null_values_per_column)
```

39717 rows × 1 columns

3.2 Removing the columns with high percentage of missing values (>50%)

In [222]:
```
#removing columns where we have>50% of the values are null
threshold = 0.5
percentage_null_values = (loan_df.isnull().mean() * 100).round(2)
columns_to_remove = percentage_null_values[percentage_null_values > threshold]
loan_df = loan_df.drop(columns=columns_to_remove)
```

```
id                          0
member_id                   0
loan_amnt                   0
funded_amnt                 0
funded_amnt_inv             0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
pymnt_plan                  0
url                         0
purpose                     0
title                      11
zip_code                    0
addr_state                  0
dti                         0
delinq_2yrs                 0
earliest_cr_line            0
inq_last_6mths              0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                 50
total_acc                   0
initial_list_status         0
out_prncp                   0
out_prncp_inv               0
total_pymnt                 0
total_pymnt_inv             0
total_rec_prncp             0
total_rec_int               0
total_rec_late_fee          0
recoveries                  0
collection_recovery_fee     0
last_pymnt_d               71
last_pymnt_amnt             0
last_credit_pull_d          2
collections_12_mths_ex_med 56
policy_code                 0
application_type            0
acc_now_delinq              0
chargeoff_within_12_mths   56
delinq_amnt                 0
tax_liens                  39
dtype: int64
```

In [224]:
```
#getting the dataframe dimensions after removing columns with >90% values are
loan_df.shape
```

Out[224]: (39717, 50)

In [225]:
```
# re-checking columns with missing
round(100.0 * loan_df.isnull().sum()/len(loan_df),2).sort_values()
```

Out[225]:
```
id                          0.00
delinq_amnt                 0.00
open_acc                    0.00
pub_rec                     0.00
revol_bal                   0.00
total_acc                   0.00
initial_list_status         0.00
out_prncp                   0.00
out_prncp_inv               0.00
total_pymnt                 0.00
total_pymnt_inv             0.00
total_rec_prncp             0.00
total_rec_int               0.00
```

In [225]:
```
# re-checking columns with missing
round(100.0 * loan_df.isnull().sum()/len(loan_df),2).sort_values()
```

Out[225]:
```
id                            0.00
delinq_amnt                   0.00
open_acc                      0.00
pub_rec                       0.00
revol_bal                     0.00
total_acc                     0.00
initial_list_status           0.00
out_prncp                     0.00
out_prncp_inv                 0.00
total_pymnt                   0.00
total_pymnt_inv               0.00
total_rec_prncp               0.00
total_rec_int                 0.00
total_rec_late_fee            0.00
recoveries                    0.00
collection_recovery_fee       0.00
last_pymnt_amnt               0.00
policy_code                   0.00
application_type              0.00
acc_now_delinq                0.00
earliest_cr_line              0.00
delinq_2yrs                   0.00
inq_last_6mths                0.00
addr_state                    0.00
member_id                     0.00
loan_amnt                     0.00
funded_amnt                   0.00
funded_amnt_inv               0.00
term                          0.00
int_rate                      0.00
installment                   0.00
dti                           0.00
sub_grade                     0.00
home_ownership                0.00
grade                         0.00
verification_status           0.00
issue_d                       0.00
loan_status                   0.00
pymnt_plan                    0.00
url                           0.00
purpose                       0.00
zip_code                      0.00
annual_inc                    0.00
last_credit_pull_d            0.01
title                         0.03
tax_liens                     0.10
revol_util                    0.13
collections_12_mths_ex_med    0.14
chargeoff_within_12_mths      0.14
last_pymnt_d                  0.18
dtype: float64
```

3.3 Updating the right columns types

In [226]:
```
#Columns int_rate & revol_util have percentage values of dtype objects, so cor
loan_df['int_rate'] = loan_df['int_rate'].astype(str)
loan_df['int_rate'] = loan_df['int_rate'].str.replace('%', '').astype(float)
```

3.4 Subsetting the data to filter only the defaulters data

In [228]:
```
#creatting subset of the data for only defualted customers for further steps
loan_df_chargedoff = loan_df[loan_df['loan_status'] == 'Charged Off']
```

In [227]:
```
loan_df['issue_d'] = pd.to_datetime(loan_df.issue_d, format='%b-%y')
#subtracting values more than 2023  with 100 will give more actual time
loan_df['issue_d'] = loan_df['issue_d'].apply(lambda x: x-pd.DateOffset(years=
```

In [229]:
```
#getting the dataframe dimensions after subsetting the data to only the defaul
loan_df_chargedoff.shape
```

Out[229]:
```
(5627, 53)
```

```
#Analysis by creating derived variables and creating new columns for datetime colum
loan_df['issue_d_year'] = loan_df.issue_d.dt.year
loan_df['issue_d_month'] = loan_df.issue_d.dt.strftime('%b')
loan_df['issue_d_weekday'] = loan_df.issue_d.dt.weekday
```

In [230]:
```
#getting the column informations
loan_df['issue_d_year'] = loan_df['issue_d_year'].astype(object)
loan_df['issue_d_weekday'] = loan_df['issue_d_weekday'].astype(object)
loan_df_chargedoff.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5627 entries, 1 to 39688
```

loan_df['int_rate'] = loan_df['int_rate'].str.replace('%', '').astype(float)

3.4 Subsetting the data to filter only the defaulters data

In [228]:
```
#creatting subset of the data for only defualted customers for further steps
loan_df_Chargedoff = loan_df[loan_df['loan_status'] == 'Charged Off']
```

In [227]:
```
loan_df['issue_d'] = pd.to_datetime(loan_df.issue_d, format='%b-%y')
#subtracting values more than 2023  with 100 will give more actual time
loan_df['issue_d'] = loan_df['issue_d'].apply(lambda x: x-pd.DateOffset(years=
```

In [229]:
```
#getting the dataframe dimensions after subsetting the data to only the defaul
loan_df_Chargedoff.shape
```

Out[229]:
```
(5627, 53)
```

In [230]:
```
#Changing the datatype of variables and creating new columns for datetime colum
loan_df['issue_d_year'] = loan_df.issue_d.dt.year
loan_df['issue_d_month'] = loan_df.issue_d.dt.strftime('%b')
loan_df['issue_d_weekday'] = loan_df.issue_d.dt.weekday
loan_df['issue_d_year'] = loan_df['issue_d_year'].astype(object)
loan_df['issue_d_weekday'] = loan_df['issue_d_weekday'].astype(object)
#getting the column informations
loan_df_Chargedoff.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5627 entries, 1 to 39688
Data columns (total 53 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   id                          5627 non-null   int64
 1   member_id                   5627 non-null   int64
 2   loan_amnt                   5627 non-null   int64
 3   funded_amnt                 5627 non-null   int64
 4   funded_amnt_inv             5627 non-null   float64
 5   term                        5627 non-null   object
 6   int_rate                    5627 non-null   float64
 7   installment                 5627 non-null   float64
 8   grade                       5627 non-null   object
 9   sub_grade                   5627 non-null   object
 10  home_ownership              5627 non-null   object
 11  annual_inc                  5627 non-null   float64
 12  verification_status         5627 non-null   object
 13  issue_d                     5627 non-null   datetime64[ns]
 14  loan_status                 5627 non-null   object
 15  pymnt_plan                  5627 non-null   object
 16  url                         5627 non-null   object
 17  purpose                     5627 non-null   object
 18  title                       5625 non-null   object
 19  zip_code                    5627 non-null   object
 20  addr_state                  5627 non-null   object
 21  dti                         5627 non-null   float64
 22  delinq_2yrs                 5627 non-null   int64
 23  earliest_cr_line            5627 non-null   object
 24  inq_last_6mths              5627 non-null   int64
 25  open_acc                    5627 non-null   int64
 26  pub_rec                     5627 non-null   int64
 27  revol_bal                   5627 non-null   int64
 28  revol_util                  5611 non-null   object
 29  total_acc                   5627 non-null   int64
 30  initial_list_status         5627 non-null   object
 31  out_prncp                   5627 non-null   float64
 32  out_prncp_inv               5627 non-null   float64
 33  total_pymnt                 5627 non-null   float64
 34  total_pymnt_inv             5627 non-null   float64
 35  total_rec_prncp             5627 non-null   float64
 36  total_rec_int               5627 non-null   float64
 37  total_rec_late_fee          5627 non-null   float64
 38  recoveries                  5627 non-null   float64
 39  collection_recovery_fee     5627 non-null   float64
 40  last_pymnt_d                5556 non-null   object
 41  last_pymnt_amnt             5627 non-null   float64
 42  last_credit_pull_d          5626 non-null   object
 43  collections_12_mths_ex_med  5621 non-null   float64
 44  policy_code                 5627 non-null   int64
 45  application_type            5627 non-null   object
 46  acc_now_delinq              5627 non-null   int64
 47  chargeoff_within_12_mths    5621 non-null   float64
 48  delinq_amnt                 5627 non-null   int64
 49  tax_liens                   5626 non-null   float64
 50  issue_d_year                5627 non-null   object
 51  issue_d_month               5627 non-null   object
 52  issue_d_weekday             5627 non-null   object
dtypes: datetime64[ns](1), float64(18), int64(13), object(21)
memory usage: 2.3+ MB
```

In [231]:
```
#Creating the list of Categorical Columns
Categorical_columns = ['term','grade','sub_grade','home_ownership',
                       'purpose','addr_state']
```

In [232]:
```
#Creating the list of Numerical columns
Numerical_columns = ['loan_amnt','funded_amnt','funded_amnt_inv','installment'
                     'open_acc','pub_rec','revol_bal','total_acc','total_pymnt'
                     ,'recoveries','collection_recovery_fee','last_pymnt_amnt']
```

4 Univariate Analysis

Under univariate analysis, we will look at the percentage of distribution of values of

```
 42  last_credit_pull_d          5626 non-null   object
 43  collections_12_mths_ex_med  5621 non-null   float64
 44  policy_code                 5627 non-null   int64
 45  application_type            5627 non-null   object
 46  acc_now_delinq              5627 non-null   int64
 47  chargeoff_within_12_mths    5621 non-null   float64
 48  delinq_amnt                 5627 non-null   int64
 49  tax_liens                   5626 non-null   float64
 50  issue_d_year                5627 non-null   object
 51  issue_d_month               5627 non-null   object
 52  issue_d_weekday             5627 non-null   object
dtypes: datetime64[ns](1), float64(18), int64(13), object(21)
memory usage: 2.3+ MB
```

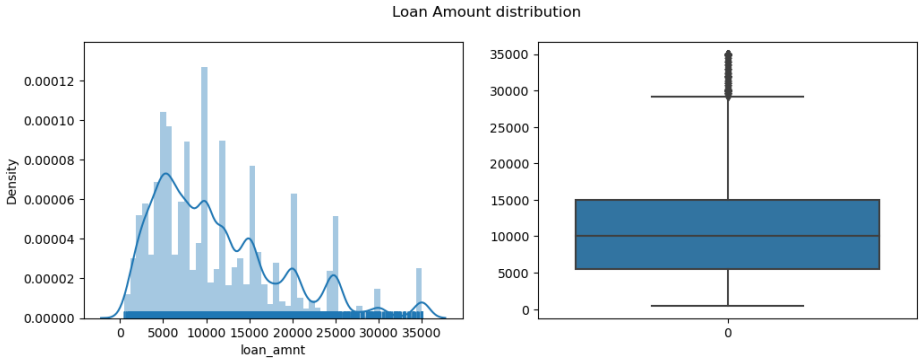
In [231]:
```
#Creating the list of Categorical Columns
Categorical_columns = ['term','grade','sub_grade','home_ownership',
                       'purpose','addr_state']
```

In [232]:
```
#Creating the List of Numerical columns
Numerical_columns = ['loan_amnt','funded_amnt','funded_amnt_inv','installment'
                     'open_acc','pub_rec','revol_bal','total_acc','total_pymnt'
                     ,'recoveries','collection_recovery_fee','last_pymnt_amnt']
```

4 Univariate Analysis

Under univariate analysis, we will look at the percentage of distribution of values of categorical variable

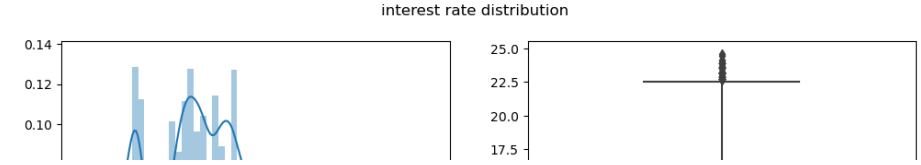
In [233]:
```
#Increasing the figure size of plot
plt.figure(figsize=(12,4))

#Setting subplot index
plt.subplot(1,2,1)

#Histogram plot
sns.distplot(a=loan_df.loan_amnt, rug=True)
plt.subplot(1,2,2)

#Box plot
sns.boxplot(data=loan_df.loan_amnt)

#Single title for both subplots.
plt.suptitle('Loan Amount distribution')
plt.show()
```



In [234]:
```
#Loan amount in percentiles
loan_df.loan_amnt.describe(percentiles=[0.05,0.1,0.25,0.5,0.75,0.9,0.95,0.99])
```

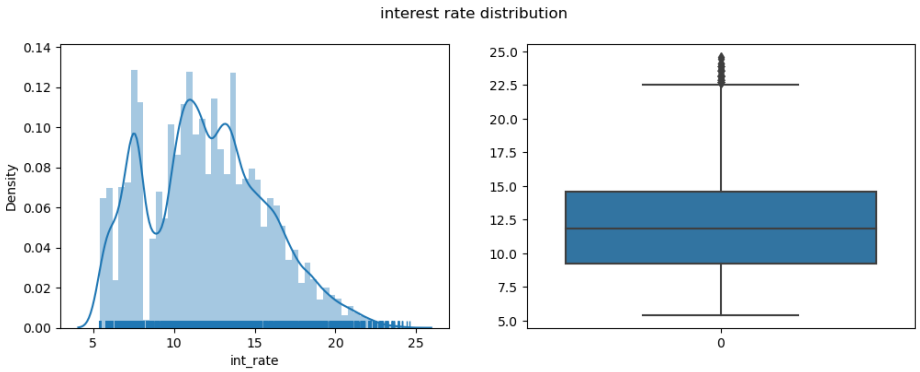
Out[234]:
```
count    39717.000000
mean     11219.443815
std       7456.670694
min        500.000000
5%        2400.000000
10%       3200.000000
25%       5500.000000
50%      10000.000000
75%      15000.000000
90%      22000.000000
95%      25000.000000
99%      35000.000000
max      35000.000000
```

In [235]:
```
#Intest Rate Analysis
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
sns.distplot(a=loan_df.int_rate, rug=True)
plt.subplot(1,2,2)
sns.boxplot(data=loan_df.int_rate)
plt.suptitle('interest rate distribution')
plt.show()
```



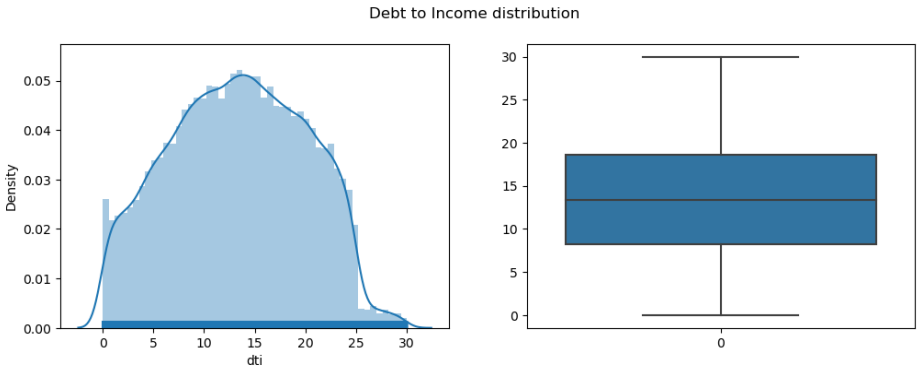
```
std       7456.670694
min        500.000000
5%        2400.000000
10%       3200.000000
25%       5500.000000
50%      10000.000000
75%      15000.000000
90%      21600.000000
95%      25000.000000
99%      35000.000000
max      35000.000000
```

In [235]:
```
#intest Rate Analysis
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
sns.distplot(a=loan_df.int_rate, rug=True)
plt.subplot(1,2,2)
sns.boxplot(data=loan_df.int_rate)
plt.suptitle('interest rate distribution')
plt.show()
```



From the above observation we can conclude that most of the intrest rate range between 9-14.5%, where some loans are issued at higher 22.5%, this can be a outlier.

In [236]:
```
##

var1 = loan_df.dti

plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
sns.distplot(a=var1, rug=True)
plt.subplot(1,2,2)
sns.boxplot(data=var1)
plt.suptitle('Debt to Income distribution')
plt.show()
```



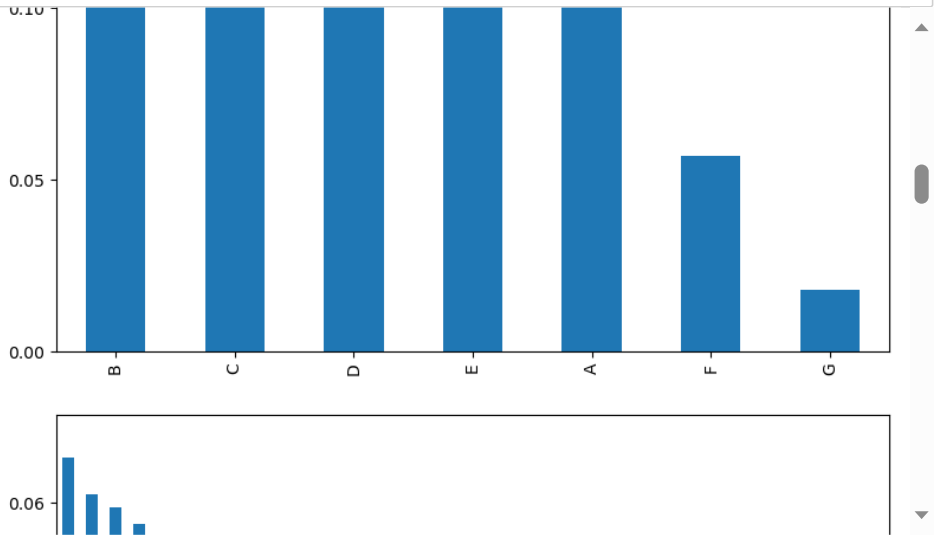
In [237]:
```
# Analysis based on the charged off cases
for i in categorical_columns :
    plt.figure(figsize = (20,10))
    plt.subplot(1,2,1)
    loan_df_Chargedoff[i].value_counts(normalize = True).plot.bar()
```

Observation: From the above distribution it looks like a normal distribution and most of the borrower's dti is less than 30

In [237]:
```
# Analysis based on the charged off cases
```

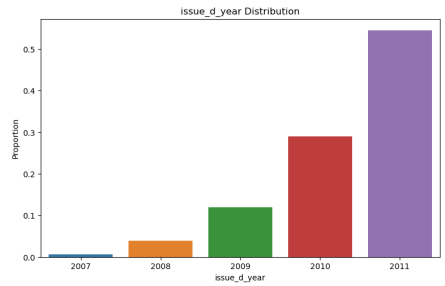
Observation: From the above distribution it looks like a normal distribution and most of the borrowers dti is less than 30

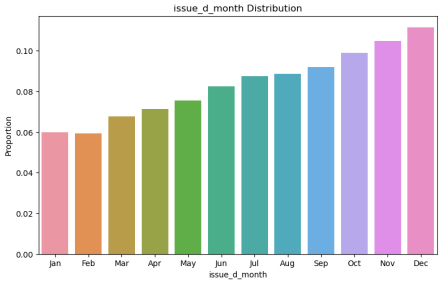
```
for i in categorical_columns :
    plt.figure(figsize = (20,10))
    plt.subplot(1,2,1)
    loan_df_Chargedoff[i].value_counts(normalize = True).plot.bar()
```

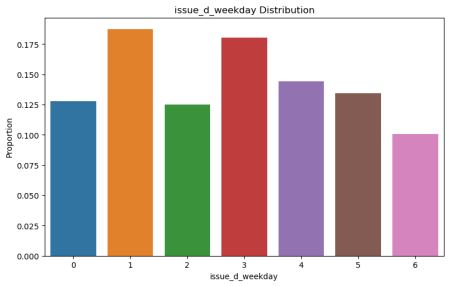


In [238]:
```
var = 'issue_d_year'
#Probability / Percentage of each values
prob_df = loan_df[var].value_counts(normalize=True).reset_index()

plt.figure(figsize=(20,12))
plt.subplot(2,2,1)
sns.barplot(x='index', y=var, data=prob_df)
plt.xlabel(var)
plt.ylabel('Proportion')
plt.title(var+' Distribution')

var = 'issue_d_month'
#Probability / Percentage of each values
month_order = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
prob_df = loan_df[var].value_counts(normalize=True).reindex(month_order).reset
plt.subplot(2,2,2)
```

In [238]:
```
var = 'issue_d_year'
#Probability / Percentage of each values
prob_df = loan_df[var].value_counts(normalize=True).reset_index()

plt.figure(figsize=(20,12))
plt.subplot(2,2,1)
sns.barplot(x='index', y=var, data=prob_df)
plt.xlabel(var)
plt.ylabel('Proportion')
plt.title(var+' Distribution')

var = 'issue_d_month'
#Probability / Percentage of each values
month_order = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
prob_df = loan_df[var].value_counts(normalize=True).reindex(month_order).reset
plt.subplot(2,2,2)
sns.barplot(x='index', y=var, data=prob_df)
plt.xlabel(var)
plt.ylabel('Proportion')
plt.title(var+' Distribution')


var = 'issue_d_weekday'
#Probability / Percentage of each values
prob_df = loan_df[var].value_counts(normalize=True).reset_index()
plt.subplot(2,2,3)
sns.barplot(x='index', y=var, data=prob_df)
plt.xlabel(var)
plt.ylabel('Proportion')
plt.title(var+' Distribution')

plt.show()
```



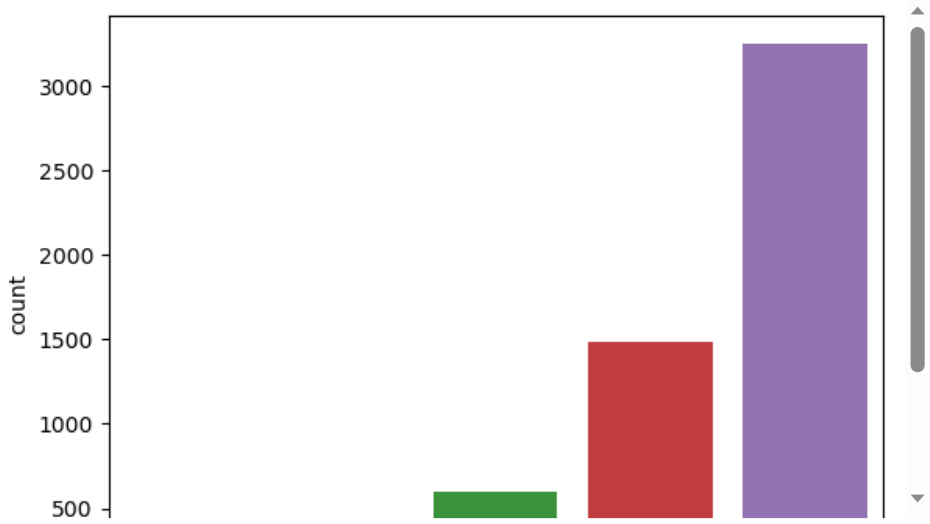




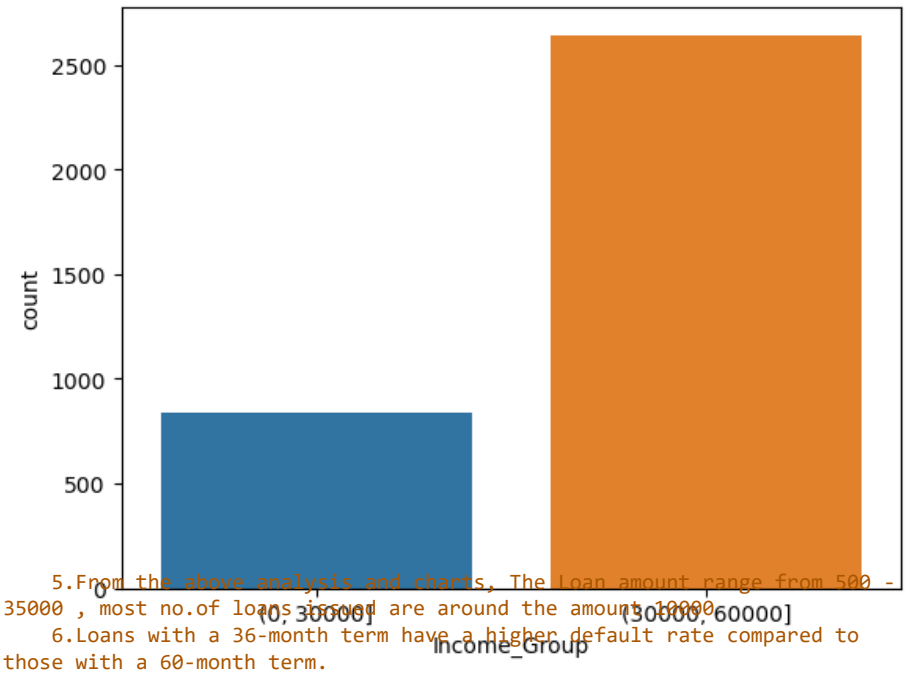
In [239]:
```
loan_df['issue_d'] = pd.to_datetime(loan_df.issue_d, format='%b-%y')
#subtracting values more than 2023  with 100 will give more actual time
loan_df['issue_d'] = loan_df['issue_d'].apply(lambda x: x-pd.DateOffset(years=
```

In [240]:
```
#converting issue_d to a date datatype
loan_df_Chargedoff["issue_d"] = pd.to_datetime(loan_df_Chargedoff["issue_d"])

#creating a new year derived column to see year on year defaulters trend
loan_df_Chargedoff["issue_d_Year"]= loan_df_Chargedoff["issue_d"] .dt.year

#creating a plot to see the Year on Year trend on the defaulters
sns.countplot(data = loan_df_Chargedoff, x ="issue_d_Year")
plt.show()
```

In [240]:
```
#converting issue_d to a date datatype
loan_df_Chargedoff["issue_d"] = pd.to_datetime(loan_df_Chargedoff["issue_d"])

#creating a new year derived column to see year on year defaulters trend
loan_df_Chargedoff["issue_d_Year"]= loan_df_Chargedoff["issue_d"] .dt.year

#creating a plot to see the Year on Year trend on the defaulters
sns.countplot(data = loan_df_Chargedoff, x ="issue_d_Year")
plt.show()
```



In [241]:
```
#creating a derived column to grouop the income groups
bins = [0,30000,60000]
lables = ['<30000','>30000',float('inf')]
loan_df_Chargedoff["Income_Group"] = pd.cut(loan_df_Chargedoff["annual_inc"],

#creating a countplot to visulize the income group defaulters
sns.countplot(data = loan_df_Chargedoff, x ="Income_Group")
plt.show()
```



5.From the above analysis and charts, The Loan amount range from 500 - 35000 , most no.of loans issued are around the amount 10000

6.Loans with a 36-month term have a higher default rate compared to those with a 60-month term.

7.Loans with grades B, C, and D have a higher default rate compared to loans with other grades.

Key Interpretation from univariate analysis of categorical variables

8.The top 5 subgrades with the highest default rates are B5, B3, C1, B4, and C2.

Important observations

9.Borrowers who own their homes have a lower default rate compared to those who do not own a home.

10.The grade C has the highest number of defaulting loans among all grades.

11.The individuals earning more than 30,000 have more paid to the default two thirds loans compared to those earning less than 30,000

3.Around 85% of the loans issued are Fully paid 15% of loans issued are defaulted

4.Lending Club issue loans only on 2 terms. 36 months and 60 months

5.From the above analysis and charts, The Loan amount range from 500 - 35000 , most no.of loans issued are around the amount 10000
6.Loans with a 36-month term have a higher default rate compared to those with a 60-month term.
7.Loans with grades B, C, and D have a higher default rate compared to loans with other grades.

Key Interpretation from univariate analysis of categorical variables

8.The top 5 subgrades with the highest default rates are B5, B3, C1, B4, and C2.

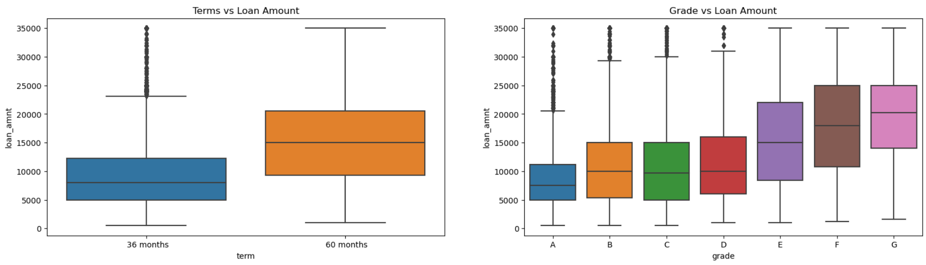
Important observations

9.Borrowers who own their homes have a lower default rate compared to those who do not own a home.
10.The state CA has the highest number of defaulting loans.
1.Lending Club has doubled its loans every year.
11.Individuals earning more than 30,000 are more inclined to default on their loans compared to those earning less than 30,000
2.The last two quarters of the year are more compared to the first two quarters.
3.Around 85% of the loans issued are Fully paid 15% of loans issued are defaulted
4.Lending Club issue loans only on 2 terms. 36 months and 60 months

Segmented Univariate Analysis

In [242]:
```python
#Loan Amount

plt.figure(figsize=(20,5))
plt.subplot(121)
sns.boxplot(x='term', y=loan_df.loan_amnt, data=loan_df)
plt.title('Terms vs Loan Amount')
plt.subplot(122)
plt.title('Grade vs Loan Amount')
#Finding grades with sorted alphabetical order
grade_ord = loan_df.grade.unique()
grade_ord.sort()
sns.boxplot(x='grade', y=loan_df.loan_amnt, order = grade_ord, data=loan_df)
```

Out[242]: <AxesSubplot:title={'center':'Grade vs Loan Amount'}, xlabel='grade', ylabel='loan_amnt'>

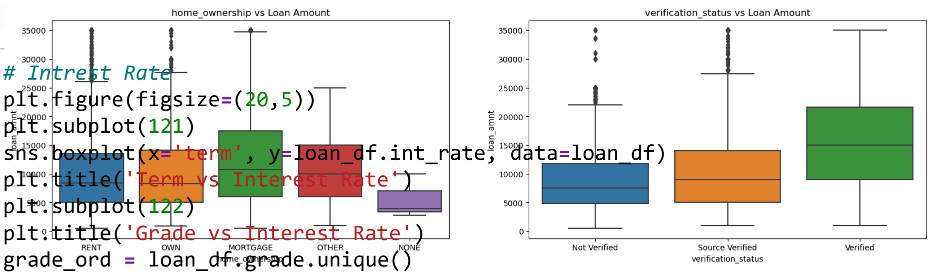


Observations: Higher amount loans have higher tenure which is 60 months

In [243]:
```python
#Home Ownership

plt.figure(figsize=(20,5))
plt.subplot(121)
sns.boxplot(x='home_ownership', y=loan_df.loan_amnt, data=loan_df)
plt.title('home_ownership vs Loan Amount')
plt.subplot(122)
plt.title('verification_status vs Loan Amount')
verification_status_ord = loan_df.verification_status.unique()
verification_status_ord.sort()
sns.boxplot(x='verification_status', y=loan_df.loan_amnt, order = verificatior
```

Out[243]: <AxesSubplot:title={'center':'verification_status vs Loan Amount'}, xlabel='verification_status', ylabel='loan_amnt'>

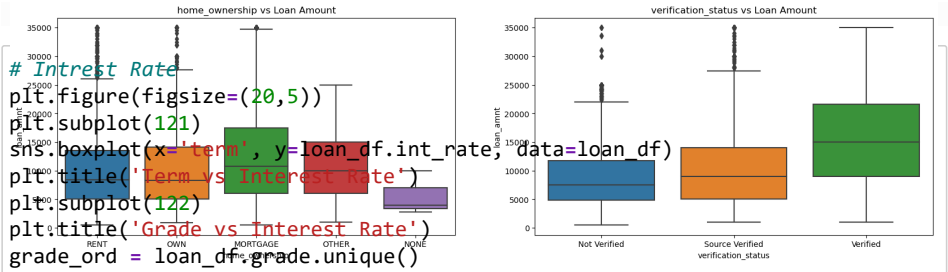
Observations: Mortgage Borrowers are higher on the IQR range and median level and most of the borrower's who has taken loan more than 9000 are verified (KYC is done



In [244]:
```python
# Intrest Rate
plt.figure(figsize=(20,5))
plt.subplot(121)
sns.boxplot(x='term', y=loan_df.int_rate, data=loan_df)
plt.title('Term vs Interest Rate')
plt.subplot(122)
plt.title('Grade vs Interest Rate')
grade_ord = loan_df.grade.unique()
grade_ord.sort()
sns.boxplot(x='grade', y=loan_df.int_rate, order = grade_ord, data=loan_df)
```

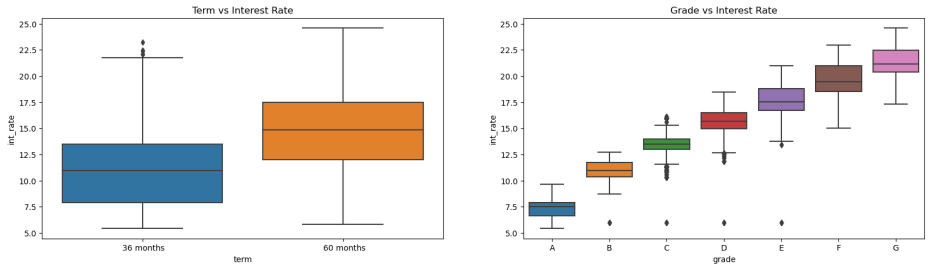
Out[243]: <AxesSubplot:title={'center':'verification_status vs Loan Amount'}, xlabel
= 'verification_status', ylabel='loan_amnt'>

Observations: Mortgage Borrowers are higher on the IQR range and median level and most of the borrower's who has taken loan more than 9000 are verified (KYC is done



In [244]:
```
# Intrest Rate
plt.figure(figsize=(20,5))
plt.subplot(121)
sns.boxplot(x='term', y=loan_df.int_rate, data=loan_df)
plt.title('Term vs Interest Rate')
plt.subplot(122)
plt.title('Grade vs Interest Rate')
grade_ord = loan_df.grade.unique()
grade_ord.sort()
sns.boxplot(x='grade', y=loan_df.int_rate, order = grade_ord, data=loan_df)
```

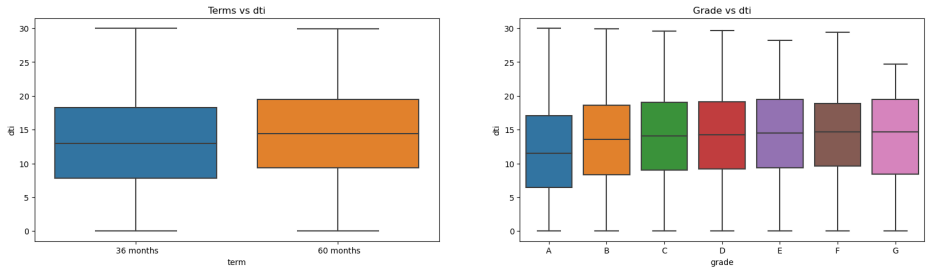
Out[244]: <AxesSubplot:title={'center':'Grade vs Interest Rate'}, xlabel='grade', ylab
el='int_rate'>



Observations: The intrest rates are higher for higher tenure loans and grades from 'A' to 'G' have higher intrest rates.

In [245]:
```
## DTI

plt.figure(figsize=(20,5))
plt.subplot(121)
sns.boxplot(x='term', y=loan_df.dti, data=loan_df)
plt.title('Terms vs dti')
plt.subplot(122)
plt.title('Grade vs dti')
grade_ord = loan_df.grade.unique()
grade_ord.sort()
sns.boxplot(x='grade', y=loan_df.dti, order = grade_ord, data=loan_df)
```

Out[245]: <AxesSubplot:title={'center':'Grade vs dti'}, xlabel='grade', ylabel='dti'>



Observations: DTi is higher for peopie who opted for higher tenure - 60 months and 'A' grade borrowers are having low DTI than other grades. For higher repayment percentage DTI should be low

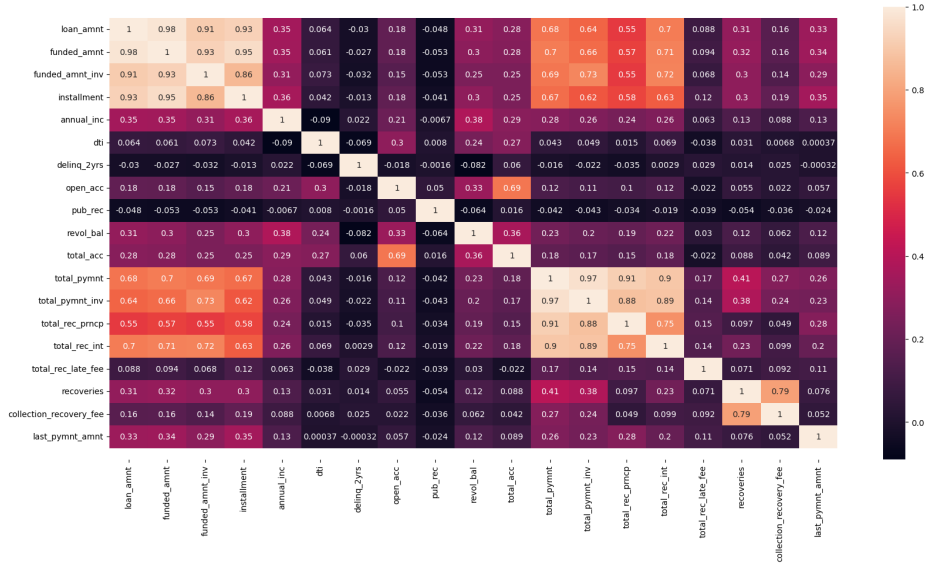
In [246]:
```
plt.figure(figsize=(20,10))
sns.heatmap(loan_df_Chargedoff[Numerical_columns].corr(),annot = True)
b, t = plt.ylim()
b += 0.5
t -= 0.5
plt.ylim(b, t)
plt.yticks(rotation = 0)
plt.show()
```

Correlation for numerical columns

Observations: DTI is higher for people who opted for higher tenure - 60 months and 'A' grade borrowers are having low DTI than other grades. For higher repayment percentage DTI should be low

In [246]:
```
plt.figure(figsize = (20,10))
sns.heatmap(loan_df_Chargedoff[Numerical_columns].corr(),annot = True)
b, t = plt.ylim()
b += 0.5
t -= 0.5
plt.ylim(b, t)
plt.yticks(rotation = 0)
plt.show()
```

Correlation for numerical columns



Important observations

1. loan_amnt, funded_amnt, funded_amnt_inv& installment columns are highly correalted
2. total_pymmnt, total_payment_inv,total_rec_prncp,total_rec_int are moderatly correlated to point 1 columns
3. for the further analysis we could reduce the features which are highly correlated